nginx

==============================

## **1.Nginx简介**

Nginx (engine x) 是一个高性能的HTTP和反向代理web服务器，同时也提供了IMAP/POP3/SMTP服务。

其将源代码以类BSD许可证的形式发布，因它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。2011年6月1日，nginx 1.0.4发布。

Nginx是一款轻量级的Web 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，在BSD-like 协议下发行。其特点是占有内存少，并发能力强，事实上nginx的并发能力在同类型的网页服务器中表现较好，中国大陆使用nginx网站用户有：百度、京东、新浪、网易、腾讯、淘宝等。

Nginx 是一个很强大的高性能Web和反向代理服务，它具有很多非常优越的特性：  
在连接高并发的情况下，Nginx是Apache服务不错的替代品：Nginx在美国是做虚拟主机生意的老板们经常选择的软件平台之一。能够支持高达 50,000 个并发连接数的响应，感谢Nginx为我们选择了 epoll and kqueue作为开发模型。

## **2.正向代理**

在客户端(浏览器)配置代理服务器，通过代理服务器进行互联网访问。

## **3.反向代理**

我们只需要将请求发送到反向代理服务器，由反向代理服务器去选择目标服务器获取数据后，在返回给客户端，此时反向代理服务器和目标服务器对外就是一个服务器,暴露的是代理服务器地址，隐藏了真实服务器IP地址。

## **4.负载均衡**

单个服务器解决不了，我们增加服务器的数量，然后将请求分发到各个服务器上，将原先请求集中到单个服务器上的情况改为将请求分发到多个服务器上，将负载分发到不同的服务器，也就是我们所说的负载均衡。

## **5.动静分离**

为了加快网站的解析速度，可以把动态页面和静态页面由不同的服务器来解析，加快解析速度。降低原来单个服务器的压力。

## **6.请列举Nginx的一些特性？**

Nginx服务器的特性包括：反向代理/L7负载均衡器 ；嵌入式Perl解释器 ；动态二进制升级；可用于重新编写URL，具有非常好的PCRE支持。

## **7.nginx和apache的区别？**

轻量级，同样起web 服务，比apache 占用更少的内存及资源；

抗并发，nginx处理请求是异步非阻塞的，而apache 则是阻塞型的，在高并发下nginx 能保持低资源低消耗高性能；

高度模块化的设计，编写模块相对简单；最核心的区别在于apache是同步多进程模型，一个连接对应一个进程；

nginx是异步的，多个连接（万级别）可以对应一个进程。

## **8.nginx是如何实现高并发的？**

一个主进程，多个工作进程，每个工作进程可以处理多个请求，每进来一个request，会有一个worker进程去处理。但不是全程的处理，处理到可能发生阻塞的地方，比如向上游（后端）服务器转发request，并等待请求返回。那么，这个处理的worker继续处理其他请求，而一旦上游服务器返回了，就会触发这个事件，worker才会来接手，这个request才会接着往下走。由于web server的工作性质决定了每个request的大部份生命都是在网络传输中，实际上花费在server机器上的时间片不多。这是几个进程就解决高并发的秘密所在。即@skoo所说的webserver刚好属于网络io密集型应用，不算是计算密集型。

## **9.Nginx如何处理HTTP请求？**

Nginx使用反应器模式。主事件循环等待操作系统发出准备事件的信号，这样数据就可以从套接字读取，在该实例中读取到缓冲区并进行处理。单个线程可以提供数万个并发连接。

## **10.为什么要用Nginx?**

* 跨平台、配置简单
* 非阻塞、高并发连接：  
  处理2-3万并发连接数，官方监测能支持5万并发
* 内存消耗小：  
  开启10个nginx才占150M内存，Nginx采取了分阶段资源分配技术
* nginx处理静态文件好,耗费内存少
* 内置的健康检查功能：  
  如果有一个服务器宕机，会做一个健康检查，再发送的请求就不会发送到宕机的服务器了。重新将请求提交到其他的节点上。
* 节省宽带：  
  支持GZIP压缩，可以添加浏览器本地缓存
* 稳定性高：  
  宕机的概率非常小
* master/worker结构：  
  一个master进程，生成一个或者多个worker进程
* 接收用户请求是异步的：  
  浏览器将请求发送到nginx服务器，它先将用户请求全部接收下来，再一次性发送给后端web服务器，极大减轻了web服务器的压力,一边接收web服务器的返回数据，一边发送给浏览器客户端
* 网络依赖性比较低，只要ping通就可以负载均衡
* 可以有多台nginx服务器

## **11.为什么Nginx性能这么高？**

得益于它的事件处理机制：  
异步非阻塞事件处理机制：

运用了epoll模型，提供了一个队列，排队解决

## **12.为什么不使用多线程？**

Nginx:采用单线程来异步非阻塞处理请求（管理员可以配置Nginx主进程的工作进程的数量），不会为每个请求分配cpu和内存资源，节省了大量资源，同时也减少了大量的CPU的上下文切换，所以才使得Nginx支持更高的并发。

## **13.在Nginx中如何在URL中保留双斜线?**

要在URL中保留双斜线，就必须使用merge\_slashes\_off；语法:merge\_slashes [on/off] ； 默认值: merge\_slashes on ；环境: http，server

## **14.ngx\_http\_upstream\_module的作用是什么?**

ngx\_http\_upstream\_module用于定义可通过fastcgi传递、proxy传递、uwsgi传递、memcached传递和scgi传递指令来引用的服务器组。

## **15.什么是C10K问题?**

C10K问题是指无法同时处理大量客户端(10,000)的网络套接字。

## **16.请陈述stub\_status和sub\_filter指令的作用是什么?**

* Stub\_status指令：该指令用于了解Nginx当前状态的当前状态，如当前的活动连接，接受和处理当前读/写/等待连接的总数 ；
* Sub\_filter指令：它用于搜索和替换响应中的内容，并快速修复陈旧的数据

## **17.Nginx是否支持将请求压缩到上游?**

可以使用Nginx模块gunzip将请求压缩到上游。gunzip模块是一个过滤器，它可以对不支持“gzip”编码方法的客户机或服务器使用“内容编码:gzip”来解压缩响应。

## **18.解释如何在Nginx中获得当前的时间?**

要获得Nginx的当前时间，必须使用SSI模块、d a t e g m t 和 date\_gmt和*dateg*​*mt*和date\_local的变量。Proxy\_set\_header THE-TIME $date\_gmt;

## **19.用Nginx服务器解释-s的目的是什么?**

用于运行Nginx -s参数的可执行文件。

## **20.解释如何在Nginx服务器上添加模块?**

在编译过程中，必须选择Nginx模块，因为Nginx不支持模块的运行时间选择。

## **21.列举Nginx服务器的最佳用途。**

Nginx服务器的最佳用法是在网络上部署动态HTTP内容，使用SCGI、WSGI应用程序服务器、用于脚本的FastCGI处理程序。它还可以作为负载均衡器。

## **22.Nginx服务器上的Master和Worker进程分别是什么?**

* Master进程：读取及评估配置和维持 ；
* Worker进程：处理请求。

1. **Nginx是如何处理一个请求的呢？**

首先，nginx在启动时，会解析配置文件，得到需要监听的端口与ip地址，然后在nginx的master进程里面先初始化好这个监控的socket，再进行listen。  
然后再fork出多个子进程出来, 子进程会竞争accept新的连接。  
此时，客户端就可以向nginx发起连接了。当客户端与nginx进行三次握手，与nginx建立好一个连接后，此时，某一个子进程会accept成功，然后创建nginx对连接的封装，即ngx\_connection\_t结构体接着，根据事件调用相应的事件处理模块，如http模块与客户端进行数据的交换，最后，nginx或客户端来主动关掉连接，

1. **Nginx跨域配置**

//打开nginx的配置文件  
vim nginx.conf  
//在指定的代理位置进行设置如下代码

location ^/home {

add\_headeer ‘Access\_Control\_Allow\_Origin’ ’\*.huawei.com‘ //控制的域名范围

add\_header 'Access\_Control\_Allow\_Credentials' 'true'

add\_headeer ‘Access\_Control\_Allow\_Method’ ’GET，POST，PUT，DELETE，HEAD，OPTIONS‘ //控制的请求方法

}

**#设置允许全局跨域**

server {  
　　 ....

add\_header Access-Control-Allow-Origin \*;

add\_header Access-Control-Allow-Methods 'GET,POST';

add\_header Access-Control-Allow-Headers 'DNT,X-Mx-ReqToken,Keep-Alive,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Authorization';

}

**另一种方法，**使用Nginx转发请求。把跨域的接口写成调本域的接口，然后将这些接口转发到真正的请求地址。

把默认的server配置注释掉。

在下面增加：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | server{    listen 8888;    server\_name 192.168.1.100;      location /{      proxy\_pass http://192.168.1.100:8080;    }      location /api{      proxy\_pass http://ni.hao.sao/api;    }  } |
|  |  |

vue

参考链接：

<https://blog.csdn.net/weixin_41883384/article/details/106550364?utm_medium=distribute.pc_relevant.none-task-blog-baidulandingword-2&spm=1001.2101.3001.4242>

<https://blog.csdn.net/qq_41646249/article/details/104644647?utm_medium=distribute.pc_relevant.none-task-blog-searchFromBaidu-3.control&depth_1-utm_source=distribute.pc_relevant.none-task-blog-searchFromBaidu-3.control>

<https://blog.csdn.net/xx080/article/details/105566109?utm_medium=distribute.pc_relevant.none-task-blog-searchFromBaidu-2.control&depth_1-utm_source=distribute.pc_relevant.none-task-blog-searchFromBaidu-2.control>

虚拟dom:

<https://www.cnblogs.com/gxp69/p/11325381.html>

key:

<https://blog.csdn.net/xumaozeng/article/details/109415316?utm_medium=distribute.pc_relevant.none-task-blog-baidulandingword-3&spm=1001.2101.3001.4242>

====

**[在vue组件中访问vuex模块中的getters/action/state](https://www.cnblogs.com/caimuguodexiaohongmao/p/11854228.html)**

<https://www.cnblogs.com/caimuguodexiaohongmao/p/11854228.html>

[Vue—组件传值及vuex的使用](https://www.cnblogs.com/li-sir/p/11463195.html)

<https://www.cnblogs.com/li-sir/p/11463195.html>

1. **虚拟dom和diff算法**

虚拟dom：使用原生JS对象去描述一个DOM节点。Vue会把模板通过渲染函数变成以Vnode形式虚拟dom节点树，然后判断是否是第一次渲染，不是的话就和上一次的虚拟dom树进行比较，只渲染差异的部分

**diff算法是通过同层的树结点比较而非对树进行逐层遍历的方式，时间复杂度为O(n)**

1. **为什么需要虚拟DOM，它有什么好处?**

Web界面由DOM树(树的意思是数据结构)来构建，当其中一部分发生变化时，其实就是对应某个DOM节点发生了变化，

        虚拟DOM就是为了**解决浏览器性能问题**而被设计出来的。**如前**，若一次操作中有10次更新DOM的动作，虚拟DOM不会立即操作DOM，而是将这10次更新的diff内容保存到本地一个JS对象中，最终将这个JS对象一次性attch到DOM树上，再进行后续操作，避免大量无谓的计算量。**所以，**用JS对象模拟DOM节点的好处是，页面的更新可以先全部反映在JS对象(虚拟DOM)上，操作内存中的JS对象的速度显然要更快，等更新完成后，再将最终的JS对象映射成真实的DOM，交由浏览器去绘制。

1. **Vue key的作用**

v-if中用 key 管理可复用的元素.

v-for中需要使用key来给每个节点做一个唯一标识，**Diff算法**就可以正确的识别此节点，找到正确的位置区插入新的节点.

vue和react都实现了一套虚拟DOM，使我们可以不直接操作DOM元素，只操作数据便可以重新渲染页面。而隐藏在背后的原理便是其高效的Diff算法。

vue和react的虚拟DOM的Diff算法大致相同，其核心是基于两个简单的假设：

**1. 两个相同的组件产生类似的DOM结构，不同的组件产生不同的DOM结构。**

**2. 同一层级的一组节点，他们可以通过唯一的id进行区分**

key的作用主要是用在Vue的虚拟dom算法中，在新旧nodes比较时，如果不使用key，Vue会是使用一种最大限度减少动态元素和并且尽可能的尝试就地修改和复用相同类型的元素  
而使用key后，它会基于key的变化重新排列元素顺序，并且会移除key不存在的元素

1. **说下Vue的双向绑定**

Vue2中，使用数据劫持结合发布订阅的模式来对数据进行响应式处理，使用Object.defineProperty(obj,prop,descriptor)方法对各个属性的set，get进行劫持，在数据变动时时发布消息给订阅者，触发相应的监听回调  
在初始化Vue后，会执行$mount方法，通过mountComponent函数new一个Watcher实例，会把当前的watcher实例赋值给Dep.targer全局属性，defineReactive方法里new一个Dep类，get方法里在Dep.targer存在时把其添加到Dep类中的subs数组中，set方法当有数据变化时调用dep.notify()方法，这个方法会循环subs数组，调用每个watcher实例的update方法

vue3 使用proxy(data,handle) 来对set get进行劫持

**vue.js 是采用数据劫持结合发布者-订阅者模式的方式，通过 Object.defineProperty()来劫持各个属性的 setter，getter，在数据变动时发布消息给订阅者，触发相应的监听回调。**

**具体步骤：**  
第一步：需要 observe 的数据对象进行递归遍历，包括子属性对象的属性，都加上 setter 和 getter 这样的话，给这个对象的某个值赋值，就会触发 setter，那么就能监听到了数据变化

第二步：compile 解析模板指令，将模板中的变量替换成数据，然后初始化渲染页面视图，并将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图

第三步：Watcher 订阅者是 Observer 和 Compile 之间通信的桥梁，主要做的事情是:

* 在自身实例化时往属性订阅器(dep)里面添加自己
* 自身必须有一个 update()方法
* 待属性变动 dep.notice()通知时，能调用自身的 update() 方法，并触发 Compile 中绑定的回调，则功成身退。

第四步：MVVM 作为数据绑定的入口，整合 Observer、Compile 和 Watcher 三者，通过 Observer 来监听自己的 model 数据变化，通过 Compile 来解析编译模板指令，最终利用 Watcher 搭起 Observer 和 Compile 之间的通信桥梁，达到数据变化 -> 视图更新；视图交互变化(input) -> 数据 model 变更的双向绑定效果。

1. **Proxy 相比于 defineProperty 的优势**

**数组变化也能监听到**

**不需要深度遍历监听**

let data = { a: 1 }

let reactiveData = new Proxy(data, {

get: function(target, name){ // ... }, // ...

})

## **对MVVM的理解**

MVVM是Model、View、ViewModel三者的缩写；

* Model：代表数据模型，数据修改和业务逻辑都在Model层中定义；
* View：代表UI视图，负责数据展示；
* ViewModel：就是与界面(view)对应的Model。因为，数据库结构往往是不能直接跟界面控件一一对应上的，所以，需要再定义一个数据对象专门对应view上的控件。而ViewModel的职责就是把model对象封装成可以显示和接受输入的界面数据对象。  
  ViewModel就是View与Model的连接器，View与Model通过ViewModel实现双向绑定。

## 6.**mvvm 和 mvc 区别？**

mvc 和 mvvm 其实区别并不大。都是一种设计思想。主要就是 mvc 中 Controller 演变成 mvvm 中的 viewModel**。mvvm 主要解决了 mvc 中大量的 DOM 操作使页面渲染性能降低，加载速度变慢，影响用户体验。**和当 Model 频繁发生变化，开发者需要主动更新到 View 。

## 7.**vue生命周期的理解？**

总共分为 8 个阶段创建前/后，载入前/后，更新前/后，销毁前/后。

* 创建前/后： 在 beforeCreate 阶段，vue 实例的挂载，元素 el 还没有。
* 载入前/后：在 beforeMount 阶段，vue 实例的$el 和 data 都初始化了，但还是挂载之前为虚拟的 dom 节点，data.message 还未替换。在 mounted 阶段，vue 实例挂载完成，data.message 成功渲染。
* 更新前/后：当 data 变化时，会触发 beforeUpdate 和 updated 方法。
* 销毁前/后：在执行 destroy 方法后，对 data 的改变不会再触发周期函数，说明此时 vue 实例已经解除了事件监听以及和 dom 的绑定，但是 dom 结构依然存在

（1）beforeCreat  
第一个生命周期函数，表示实例在完全创建出来之前会执行它，在执行它时，data和methods中的数据都还未初始化。

（2）created  
第二个生命周期函数，此时data和methods已经初始化完成，created()完成之后，vue开始编译模板，最终在内存中生成一个编译好的最终模板字符串，然后把模板字符串渲染为内存中的dom

（3）beforeMount  
第三个生命周期函数，表示模板在内存中已经编译好了，但是并没有渲染到页面中。页面显示的还仅仅是模板字符串

（4）Mounted  
第四个生命周期函数，此时内存中的模板已经真实的挂载到了页面中，用户可以看到渲染好的页面了

运行中的两个事件

（5）beforeUpdated()  
执行它时，data中的数据已经被更新了，但是页面中的data还未被替换过来

（6）updated()  
执行它时，页面和data中的数据已经同步了

**详解：**

**init**

initLifecycle/Event ，往vm上挂载各种属性

callHook: beforeCreated : 实例刚创建

initInjection/initState : 初始化注入和 data 响应性

**created**: 创建完成，属性已经绑定， 但还未生成真实 dom`

进行元素的挂载： $el / vm.$mount()

是否有 template : 解析成 render function

\*.vue 文件: vue-loader 会将 <template> 编译成 render function

**beforeMount** : 模板编译/挂载之前

执行 render function ，生成真实的 dom ，并替换到 dom tree 中

**mounted :** 组件已挂载

**update**

执行 diff 算法，比对改变是否需要触发 UI 更新

flushScheduleQueue

**watcher.before : 触发 beforeUpdate 钩子** - **watcher.run() : 执行** **watcher 中的**

**notify ，通知所有依赖项更新UI**

触发 updated 钩子: 组件已更新

actived / deactivated(keep-alive) : 不销毁，缓存，组件激活与失活

**destroy**

**beforeDestroy :** 销毁开始

**销毁自身且递归销毁子组件以及事件监听**

remove() : 删除节点

watcher.teardown() : 清空依赖

vm.$off() : 解绑监听

destroyed : 完成后触发钩子

## 8.**Vue与Angular以及React的三者之间的区别**

****与AngularJS的区别：****  
相同点：都支持指令（内置指令和自定义指令）；都支持过滤器（内置过滤器和自定义过滤器）；都支持双向绑定；都不支持地段浏览器。  
不同点：**AngularJS的学习成本高**；而Vue.js本身提供的API都比较简单；

**在性能上，AngularJS以来对数据做账检查，**所以Watcher越多越慢；Vue.js使用基于依赖追踪的观察并且使用异步队列更新，所有的数据都是独立触发的。  
****与React的区别：****  
相同点：**React采用特殊的JSX语法，Vue.js在组建开发中也推崇写.vue特殊文件格式，对文件内容都有一些约定，两者都需要编译后使用**；中心思想相同：一切都是组件，组件实例之间可以嵌套；都提供合理的钩子函数，可以让开发者定制化地去处理需求；都不内置列数AJAX，Route等功能到核心包，而是以插件的方式加载；在组件开发中都支持mixins的特性。  
不同点：**React采用的Virtual DOM会对渲染出来的结果做脏检查；Vue.js在模板中提供了指令，过滤器等，可以非常方便，快捷地操作Virtual DOM。**

## **9.vue等单页面应用及其优缺点**

****单页面应用（SPA）****  
单页面应用（SPA），通俗一点说就是指只有一个主页面的应用，浏览器一开始要加载所有必须的 html, js, css。所有的页面内容都包含在这个所谓的主页面中。

****多页面应用（MPA）****  
多页面（MPA），就是指一个应用中有多个页面，页面跳转时是整页刷新。

****单页面的优点：****  
用户体验好，快，内容的改变不需要重新加载整个页面，基于这一点spa对服务器压力较小。  
前后端分离。  
页面效果会比较炫酷（比如切换页面内容时的专场动画）。  
****单页面缺点：****  
不利于seo。  
导航不可用，如果一定要导航需要自行实现前进、后退。（由于是单页面不能用浏览器的前进后退功能，所以需要自己建立堆栈管理）。  
初次加载时耗时多。  
页面复杂度提高很多。

**优点：  
Vue 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件，核心是一个响应的数据绑定系统。**

数据驱动  
组件化  
轻量  
简洁  
高效  
模块友好  
页面切换快  
**缺点：  
不支持低版本的浏览器，最低只支持到IE9；**  
**不利于SEO的优化**（如果要支持SEO，建议通过服务端来进行渲染组件）；  
**首屏加载耗时相对长一些；**  
**不可以使用浏览器的导航按钮需要自行实现前进、后退。**

## 10.**怎么定义vue-router的动态路由？获取传值？**

定义：在router目录下的index.js文件中，对path属性增加/:id  
获取：使用route对象的params.id

## 11.**vue如何获取并操作dom对象**

在标签上添加ref=‘myBox’，然后在代码中this.$refs.myBox

<template>

<div ref="myBox">

<h1>首页标题</h1>

</div></template><script>

export default {

name: "Home",

mounted(){

this.$refs.myBox.style.color = 'red'

}

}</script><style scoped>

div{

background: pink;

}</style>

## **12.vue-loader是什么？使用它的途径有哪些？**

****vue-loader作用：****  
**解析和转换.vue文件。提取出其中的逻辑代码** script,样式代码style,以及HTML 模板template，再**分别把他们交给对应的loader去处理  
**用途:****  
**js可以写es6,style样式可以写scss或less、template可以加jade等**

## 13.**vue组件间传递方法**

**props**:子组件的模板内并不能直接使用父组件的数据，所以子组件需要用props选项来获取父组件的数据

1. 动态语法：用v-bind将动态props绑定到父组件的数据，父组件的数据发生改变，子组件也会随之改变
2. 绑定修饰符：props默认为单向绑定，是为了防止子组件无意间修改父组件的状态。

.snc:双向绑定 .once:单次绑定

<div id="app">

<input type="text" v-model="info.name"/>

<child :msg.once="info"></child> //单次绑定

</div>

<script>

window.onload=function(){

new Vue({

el:'#app',

data:function(){

return {

info:{

name:"顺风车" //父组件

}

}

},

components:{

child:{

props:['msg'],

template:"<div>{{msg.name}}欢迎！</div>" //子组件

}

}

})

}

**$emit**

**bus**

**vux**

## **vue如何定义并使用过滤器？**

**定义**

new Vue({

　　　　filters:{

　　　　　　myCurrency:function(myInput){

　　　　　　　　return 处理后的数据

　　　　　　}

　　　　}

　　})

## 15.**v-if和v-show的区别**

****v-if****  
它是惰性的，只有值为真的时候才会被选渲染。  
如果从值 从 true 改变 为 false 则元素就会被销毁。  
如果又从 false 改变 true 则元素会被重建。

****v-show****  
它不管初始条件是什么，元素总是会被渲染。  
它的值的变化，只是对 css display 属性的切换。

****那么如何选择使用 v-if和v-show呢？****  
频繁切换的元素使用v-show。  
条件很少改变，则使用 v-if 较好。

## 16.**vue中常用的指令：**

****循环:v-for****  
v-for的使用，除了item属性，还有一些其他辅助属性。假设你现在循环的对象名字为students:

如果students是数组，还有index属性，如v-for="（item,index） in students";  
如果students是对象，还有value,key属性，如v-for="（value,key,index） in students";

****条件渲染****  
第一种：if … else if …else型  
是否插入元素

第二种：v-show  
是否隐藏元素

****属性绑定:v-bind****  
v-bind:属性名=“常量 || 变量名”，简写形式——:属性名=“常量 || 变量名”

****双向绑定:v-model****

所谓的双向绑定，就是你在视图层 里面改变了值，vue里面对应的值也会改变。只能给具备value属性的元素进行双向数据绑定。

## 17.**vue常用事件修饰符：**

****.stop 防止事件冒泡****  
冒泡事件：嵌套两三层父子关系，然后所有都有点击事件，点击子节点，就会触发从内至外 子节点-》父节点的点击事件

****.prevent取消默认事件****  
.prevent等同于JavaScript的event.preventDefault()，用于取消默认事件。比如我们页面的<a href="#">标签，当用户点击时，通常在浏览器的网址列出#：

****.capture 捕获事件****  
捕获事件：嵌套两三层父子关系，然后所有都有点击事件，点击子节点，就会触发从外至内 父节点-》子节点的点击事件

## **v-on可以绑定多个方法吗**

可以。如下

<input

type="text"

v-on="{

input:onInput,

focus:onFocus,

blur:onBlur,

}"

>

## 19.**对keep-alive 的了解**

是**Vue的内置组件**，能**在组件切换过程中将状态保留在内存中，防止重复渲染DOM**。

包裹动态组件时，会缓存不活动的组件实例，而不是销毁它们。

只是一个抽象组件，它不会在DOM树中渲染(真实或者虚拟都不会)，也不在父组件链中存在，比如：你永远在 this.$parent 中找不到 keep-alive 。

**keep-alive生命周期钩子函数：activated、deactivated**

**使用会将数据保留在内存中，如果要在每次进入页面的时候获取最新的数据，需要在activated阶段获取数据，承担原来created钩子中获取数据的任务。**

被包含在 中创建的组件，会多出两个生命周期的钩子: activated 与 deactivated

**activated：在组件被激活时调用，在组件第一次渲染时也会被调用，之后每次keep-alive激活时被调用。**

deactivated：在组件被停用时调用。

注意：**只有组件被 keep-alive 包裹时，这两个生命周期才会被调用，如果作为正常组件使用，是不会被调用，以及在 2.1.0 版本之后，使用 exclude 排除之后，就算被包裹在 keep-alive 中，这两个钩子依然不会被调用！另外在服务端渲染时此钩子也不会被调用的。**

****什么时候获取数据？****

**当引入keep-alive 的时候，页面第一次进入，钩子的触发顺序created-> mounted-> activated，退出时触发deactivated。当再次进入（前进或者后退）时，只触发activated。**

## 20.**route和router的区别？**

**route是“路由信息对象”，**包括path，params，hash，query，fullPath，matched，name等路由信息参数。 **而router是“路由实例”对象**包括了路由的跳转方法，钩子函数等。

## 21.**$nextTick的使用**

Vue 中的 **nextTick 涉及到 Vue** 中 **DOM 的异步更新**

Vue 实现响应式并不是数据发生变化之后 DOM 立即变化，而是按一定的策略进行 DOM 的更新。

**$nextTick 是在下次 DOM 更新循环结束之后执行延迟回调，在修改数据之后使用 $nextTick，则可以在回调中获取更新后的 DOM**

## **active-class 是哪个组件的属性？/切换路由选中样式怎么做**

actve-class是vue-router模块的router-link组件中的属性，用来做**选中样式的切换**；

详细如下：

1.active-class的作用

active-class是vue-router模块的router-link组件中的属性，用来做选中样式的切换；

2.在vue-router中要使用active-class有两种方法：

a.直接在路由js文件中配置linkActiveClass

export default new Router({

linkActiveClass: 'active'})

b.在router-link中写入active-class

<router-link to="/home" class="menu-home" active-class="active">首页</router-link>

3.如果使用第二种方法添加active-class，跳转到my页面后，两个router-link始终都会有选中样式，代码如下

<div class="menu-btn">

<router-link to="/" class="menu-home" active-class="active">

首页

</router-link></div><div class="menu-btn">

<router-link to="/my" class="menu-my" active-class="active">

我的

</router-link></div>

可能是因为 to="/" 引起的，active-class选择样式时根据路由中的路径去匹配，然后显示，例如在my页面中，路由为localhost:8080/#/my,那么to="/”和to="/my"都可以匹配到，所有都会激活选中样式

4.要解决问题也有两种方式，都是通过加入一个exact属性

a.直接在路由js文件中配置linkActiveClass

export default new Router({

linkExactActiveClass: 'active',})

b.在router-link中写入exact

<router-link to="/" class="menu-home" active-class="active" exact>首页</router-link>

5.也可以如此解决

<router-link to="/home" class="menu-home" active-class="active" exact>首页</router-link>

路由中加入重定向{

path: '/',

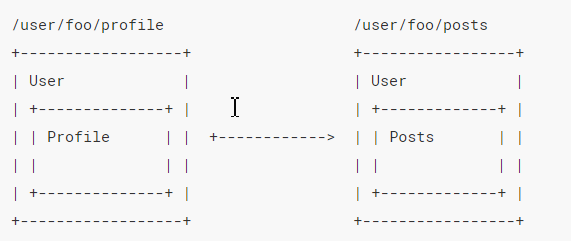
redirect: '/home'}

## **23.嵌套路由怎么定义？**

在 VueRouter 的参数中使用 children 配置，这样就可以很好的实现路由嵌套

点击不同的按钮时，他们的router-link分别所指向的组件就会被渲染到这个<router-view>中。

2、官网是怎么介绍的？



　　 这就是实际生活中的一个很好的应用界面， 通常是由多层嵌套的组件组合而成。 同样的， URL中各段动态路径也按照某种结构对应嵌套的各层组件。如上所示。

　　即user表示用户页， 而user就可以看成是vue中的一个单页面，对于一个user，一定要有哪一个用户，这里的foo（小明、小红）就代表了一个用户，这里的profile可以理解为个人主页，这里的posts可以理解为这个人所发表的文章， 而title可能是不变的，比如无论切换到这个人发表的文章，还是切换到这个人的个人主页，我们都希望在最上方显示同样的东西，而在切换的时候换的就是下面的部分，这个部分我们就可以用<router-view>来写，那么，这就是嵌套路由。

　　借助vue-router，使用嵌套路由配置，就可以很简单的表达这种关系。

<div id="app">

<router-view></router-view>

</div>

const User = {

template: `

<div class="user">

<h2>User {{ $route.params.id }}</h2>

<router-view></router-view>

</div>

`

}

const router = new VueRouter({

routes: [

{ path: '/user/:id', component: User,

children: [

{

// 当 /user/:id/profile 匹配成功，

// UserProfile 会被渲染在 User 的 <router-view> 中

path: 'profile',

component: UserProfile

},

{

// 当 /user/:id/posts 匹配成功

// UserPosts 会被渲染在 User 的 <router-view> 中

path: 'posts',

component: UserPosts

}，

// 当 /user/:id 匹配成功，

// UserHome 会被渲染在 User 的 <router-view> 中

**{**

**path: '',**

**component: UserHome**

**},**

// ...其他子路由 ]

}

]

})

　　这大致就是嵌套路由了！ 其中，第一段代码是在文件中放了<router-view>， 这是最顶层的出口，渲染**高级路由匹配到的组件。**

　　从路由配置中，我们可以看到： 当路径是 **/user/小明** 或者 **/user/小红** 时， 就会把User渲染到页面中，这个渲染的就是顶级路由。 （:id就是小明、小红，）。

　　然后这个页面中就有一个嵌套路由，当然，如果url就是 /user/小明的话，那么这个二级路由是什么都不会显示的， 如果我们希望即使只是 /user/小明， 也得渲染点什么，那么我们就可以设置 **path: "" 所对应的路由**，这样，即使 /user/小明 也是可以显示更多的了。

　　如果我们想看小明的个人主页，就是 /user/小明/profile 这时就会把 UserProfile 组件渲染到这个二级路由中了。

　　如果我们想看小明发表的文章，就是 /user/小明/posts 这时就会把 UserPost 组件渲染到这个二级路由中， 这也就是二级路由了。

　　其实概念就这么多，是不是很简单呢。

**但是还需要注意下面几点：**

　　1、从上面的代码可以看出： 在组件中，我们可以通过 $route.params.id 来获取到user/小明/post 中的小明，这样，我们就可以进行个性化的设置了。

　　2、**以 / 开头的嵌套路径会被当做根路径，** 这可以让你充分的使用嵌套组件而无需设置嵌套的路径。

也就是说： 在上面的代码中，我们在children下，**写path: "profile", 就相当于拼接成了 /user/:id/profile** ，当然，我们也可以直接写为 path: "/user/:id/profile" ，最终的结果是一样的，但是我个人认为后者的表示方法可以跟清晰的观察结构。 如果嵌套层数过多，可能会出错。

　　3、注意需要匹配一个空的情况，然后匹配一个default组件，避免出现空白页的问题。

## vue-router详解、**路由之间跳转？**

**mode**

hash history

**跳转**

声明式（标签跳转） <router-link to=""></router-link>

编程式（ js 跳转） this.$router.push('index')

**占位**

<router-view></router-view>

## 25.**懒加载（按需加载路由）（常考）**

webpack 中提供了 require.ensure()来实现按需加载。以前引入路由是通过 import 这样的方式引入，改为 const 定义的方式进行引入。

* 不进行页面按需加载引入方式：

import home from '../../common/home.vue'

* 进行页面按需加载的引入方式：

const home = r => require.ensure( [], () => r (require('../../common/home.vue')))

**·ES 提出的import方法，（------最常用------）**

**const HelloWorld = ()=>import("@/components/HelloWorld")**

export default new Router({

routes: [

{

path: '/',

name: 'HelloWorld',

**component:HelloWorld**

}

]

})

## 26.**vuex 是什么？怎么使用？哪种功能场景使用它？**

vue 框架中状态管理。在 main.js 引入 store，注入。新建了一个目录 store，….. export 。场景有：单页应用中，组件之间的状态。音乐播放、登录状态、加入购物车

IMG_256

// 新建 store.js

import vue from 'vue'

import vuex form 'vuex'

vue.use(vuex)

export default new vuex.store({

//...code

modules:{}

})

//main.js

import store from './store'

...

## 27.**vuex 有哪几种属性**

有 5 种，分别是 state、getter、mutation、action、module

## 28.**vuex 的 store 特性是什么**

* vuex 就是一个仓库，仓库里放了很多对象。其中 state 就是数据源存放地，对应于一般 vue 对象里面的 data
* state 里面存放的数据是响应式的，vue 组件从 store 读取数据，若是 store 中的数据发生改变，依赖这相数据的组件也会发生更新
* **它通过 mapState 把全局的 state 和 getters 映射到当前组件的 computed 计算属性**

## **29.vuex 的 getter 特性是什么**

* getter 可以对 state 进行计算操作，它就是 store 的计算属性
* 虽然在组件内也可以做计算属性，但是 getters 可以在多给件之间复用
* 如果一个状态只在一个组件内使用，是可以不用 getters

## **vuex 的 mutation 特性是什么**

mutation 提交state的数据更改及修改state中的数据。

* action 类似于 muation, 不同在于：action 提交的是 mutation,而不是直接变更状态
* action 可以包含任意异步操作

## **30**. **action和mutations的区别**

****action的功能和mutation是类似的，都是去变更store里的state，不过action和mutation有两点不同：****

****1、action主要处理的是异步的操作，mutation必须同步执行，而action就不受这样的限制，也就是说action中我们既可以处理同步，也可以处理异步的操作****

****2、action改变状态，最后是通过提交mutation****

### **actions和mutations是类似的，不同之处在于：**

* Action提交的是Mutation，不能够直接修改state中的状态，而Mutations是可以直接修改state中状态的；
* Action是支持异步操作的，而Mutations只能是同步操作。

## **31.vue 中 ajax 请求代码应该写在组件的 methods 中还是 vuex 的 action 中**

如果请求来的数据不是要被其他组件公用，仅仅在请求的组件内使用，就不需要放入 vuex 的 state 里

如果被其他地方复用，请将请求放入 action 里，方便复用，并包装成 promise 返回

## 32.**不用 vuex 会带来什么问题**

* 可维护性会下降，你要修改数据，你得维护 多个 个地方
* 可读性下降，因为一个组件里的数据，你根本就看不出来是从哪里来的
* 增加耦合，大量的上传派发，会让耦合性大大的增加，本来 Vue 用 Component 就是为了减少耦合，现在这么用，和组件化的初衷相背

33 **[this.$store.dispatch() 与 this.$store.commit()方法的区别](https://www.cnblogs.com/qsy0021/p/13085654.html)**

**commit: 同步操作**

this.$store.commit('方法名',值)【存储】

this.$store.state.方法名【取值】

**dispatch: 异步操作**

this.$store.dispatch('方法名',值)【存储】

this.$store.getters.方法名【取值】

**当操作行为中含有异步操作，比如向后台发送请求获取数据，就需要使用action的dispatch去完成了。  
其他使用commit即可。**

其他了解：commit=>mutations,用来触发同步操作的方法。  
dispatch =>actions,用来触发异步操作的方法。

在store中注册了mutation和action，在组件中用dispatch调用action，然后action用commit调用mutation，

## 34.[在vue组件中访问vuex模块中的getters/action/state](https://www.cnblogs.com/caimuguodexiaohongmao/p/11854228.html)

### **组件中访问模块里的state**

**传统方法：**

this.$store.state['模块名']['属性名']

例如：this.$store.state.city.list。

**mapState方法：**

import { mapState } from 'vuex'

computed: {

// ... ...mapState({

list: state => state.city.list

})

}

### **在组件中访问模块里的getter**

传统方法：

this.$store.getters['模块名/属性名']

如：this.$store.getters['city/findOne']('a')

mapGetters方法:

引入：import { mapGetters } from 'vuex'



### **组件中访问模块里的**mutations

【this.$store.commit(‘模块名/method’)】

或者mapMutations使用

### **组件中访问模块里的actions**

传统方法：

// @params:opts Object or String this.$store.dispatch('模块名/属性名', opts)

 组件中使用如：this.$store.dispatch('city/add', '上海')

mapActions方法：

引入：import { mapActions } from 'vuex'



 methods中的属性名可以更改成其他任意名字，但是属性值需要和模块名，actions属性名对应。

**35.vueBus，vue组件间的通信，eventBus是挂载在哪里的？为啥就可以进行通信**

一般在中大型项目中使用vueX就可以实现数据共享了,但是一些小项目可以使eventBus来实现组件之间的通信.

#### **原理**

其主要实现场景是在兄弟组件之间的通信,引入一个新的vue实例,然后分别调用这个实例的事件触发和监听,来实现通信和参数传递.

#### **实例**

使用场景如：兄弟组件间的通信，父组件fatherCom中同时使用到childOne和childTwo两个兄弟组件，点击子组件childTwo需要另一子组件ChildOne响应；

##### **1.定义eventBus，建立eventBus.js文件,挂载到Vue实例上，暴露出去：**

import Vue from 'vue';

　　let bus = new Vue();

　　Vue.prototype.$eventBus = bus;

　　export default bus;

##### **2.使用到eventBus的兄弟组件都引入eventBus.js：**

import eventBUs from '../../utils/eventBus.js'

##### **3.子组件childone声明发布（声明）事件，通常在mounted或created中声明：**

created() {

　　　　 eventBus.$on('getTarget', target => {

　　　　　　console.log(target);

　　　　 });

　　 }

4.子组件childtwo订阅（触发）事件：

methods: {

　　　　addCart(event) {

　　　　　　eventBus.$emit('getTarget', event.target);

　　　　 }

　　 }

## 36.**Vue响应式原理Observer、Dep、Watcher理解**

**Observer---监听数据，给数据增加dep依赖**

* Observer是用来给数据添加Dep依赖。

**Observer的作用是对整个Data进行监听，在initData这个初始方法里使用observe(data),Observer类内部通过defineReactive方法劫持data的每一个属性的getter和setter。**

**Vue的响应式数据都会有一个\_\_ob\_\_作为标记，里面存放了Observer实例，防止重复绑定。**

**Observer类主要干了以下几件事：**

* **给data绑定一个\_\_ob\_\_属性，用来存放Observer实例，避免重复绑定**
* **如果data是Object, 遍历对象的每一个属性进行defineReactive绑定**
* **如果data是Array, 则需要对每一个成员进行observe。vue.js会重写Array的push、pop、shift、unshift、splice、sort、reverse这7个方法，保证之后pop/push等操作进去的对象也有进行双向绑定.**

class Observer {

constructor() {

// 响应式绑定数据通过方法

observe(this.data);

}}

export function observe (data) {

const keys = Object.keys(data);

for (let i = 0; i < keys.length; i++) {

// 将data中我们定义的每个属性进行响应式绑定

defineReactive(obj, keys[i]);

}}

**Dep ：管理依赖**

**dep主要是用来在数据更新的时候通知watchers进行更新**

* Dep是一个发布者，可以订阅多个观察者，依赖收集之后Dep中会有一个subs存放一个或多个观察者，在数据变更的时候通知所有的watcher。
* 再复习一下，Dep和Observer的关系就是Observer监听整个data，遍历data的每个属性给每个属性绑定defineReactive方法劫持getter和setter,

**在getter的时候往Dep类里塞依赖（dep.depend），**

**在setter的时候通知所有watcher进行update(dep.notify)**

### **Watcher----接受通知，回调函数异步更新视图**

watcher接受到通知之后，会通过回调函数进行更新。  
  
接下来我们要仔细看一下watcher的源码。由之前的Dep代码可知的是，watcher需要实现以下两个作用：

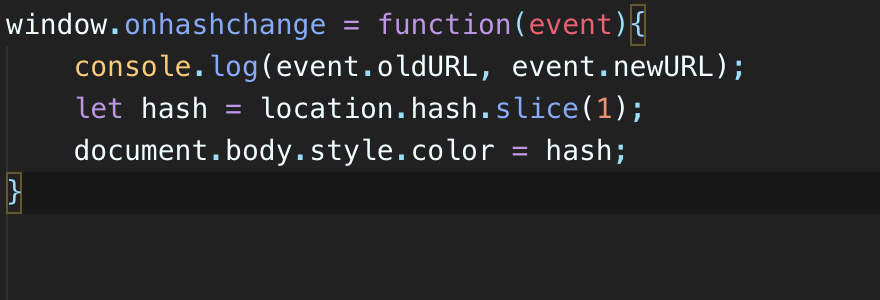
* dep.depend()的时候往dep里添加自己；
* dep.notify()的时候调用watcher.update()方法，对视图进行更新；

要注意的是，watcher中有个sync属性，绝大多数情况下，watcher并不是同步更新的，而是采用异步更新的方式，也就是调用queueWatcher(this)推送到观察者队列当中，待nextTick的时候进行调用。

## 37.[vue-router的两种模式的区别](https://www.cnblogs.com/calvin-dong/p/11222150.html)

### **hash模式**

hash模式背后的原理是onhashchange事件,可以在window对象上监听这个事件:



上面的代码可以通过改变hash来改变页面字体颜色，虽然没什么用，但是一定程度上说明了原理。

更关键的一点是，因为hash发生变化的url都会被浏览器记录下来，从而你会发现浏览器的前进后退都可以用了，同时点击后退时，页面字体颜色也会发生变化。这样一来，尽管浏览器没有请求服务器，但是页面状态和url一一关联起来，后来人们给它起了一个霸气的名字叫前端路由，成为了单页应用标配。

### **history路由**

随着history api的到来，前端路由开始进化了,前面的hashchange，你只能改变#后面的url片段，而history api则给了前端完全的自由

history api可以分为两大部分，切换和修改

#### **换历史状态**

包括back,forward,go三个方法，对应浏览器的前进，后退，跳转操作，有同学说了，(谷歌)浏览器只有前进和后退，没有跳转，嗯，在前进后退上长按鼠标，会出来所有当前窗口的历史记录，从而可以跳转(也许叫跳更合适)：

history.go(-2);//后退两次

history.go(2);//前进两次

history.back(); //后退

hsitory.forward(); //前进

通过history api，我们丢掉了丑陋的#，但是它也有个毛病：  
不怕前进，不怕后退，就怕刷新，f5，（如果后端没有准备的话）,因为刷新是实实在在地去请求服务器的,不玩虚的。

在hash模式下，前端路由修改的是#中的信息，而浏览器请求时是不带它玩的，所以没有问题.但是在history下，你可以自由的修改path，当刷新时，如果服务器中没有相应的响应或者资源，会分分钟刷出一个404来。

## **38、Vue:slot用法**

**单个 Slot**

在子组件内使用特殊的<slot>元素就可以为这个子组件添加一个 slot (插槽),在父组件模板里,插入在子组件标签内的所有内容将替代子组件的<slot>标签及它的内容.示例代码如下:

<!DOCTYPE html><html><head>

<meta charset="utf-8">

<title>示例</title>

</head><body>

<div id="app">

<child-component>

<p>分发的内容</p>

<p>更多分发的内容</p>

</child-component>

</div>

<script src="https://unpkg.com/vue/dist/vue.min.js"></script>

<script>

Vue.component('child-component', {

template: '\

<div>\

<slot>\

<p>如果父组件没用插入内容,我将作为默认出现</p>\

</slot>\

</div>'

});

var app = new Vue({

el: '#app'

})

</script>

</body></html>

子组件 child-component 的模板内定义一个 <slot> 元素,并且用一个 <p> 作为默认的内容,在父组件没有使用 slot 时,会渲染这段默认的文本;如果写入了 slot ,那就会替换整个 <slot>.所以上列渲染后的结果为:

<div id="app">

<div>

<p>分发的内容</p>

<p>更多分发的内容</p>

</div></div>

注意:子组件<slot>内的备用内容,它的作用域时子组件本身.

**具名 Slot**

给 <slot> 元素指定一个 name 后可以分发多个内容,具名 Slot 可以与单个 Slot 共存,例如下面的示例:

<!DOCTYPE html><html><head>

<meta charset="utf-8">

<title>示例</title>

</head><body>

<div id="app">

<child-component>

<h2 slot="header">标题</h2>

<p>正文内容</p>

<p>更多正文内容</p>

<div slot="footer">底部信息</div>

</child-component>

</div>

<script src="https://unpkg.com/vue/dist/vue.min.js"></script>

<script>

Vue.component('child-component', {

template: '\

<div class="component">\

<div class="header">\

<slot name="header"></slot>\

</div>\

<div class="main">\

<slot></slot>\

</div>\

<div class="footer">\

<slot name="footer"></slot>\

</div>\

</div>'

});

var app = new Vue({

el: '#app'

})

</script>

</body></html>

子组件内声明了3个 <slot> 元素,其中在<div class="main">内的<slot> 没用使用 name 特性,它将作为默认 slot 出现,父组件没有使用 slot 特性的元素与内容都将出现在这里.  
**如果没有指定默认的匿名 slot, 父组件内多余的内容片段都将被抛弃.**  
上例最终渲染后的结果为:

<div id="app">

<div class="container">

<div class="header">

<h2>标题</h2>

</div>

<div class="main">

<p>正文内容</p>

<p>更多的正文内容</p>

</div>

<div class="footer">

<div>底部信息</div>

</div>

</div>

</div>

在组合使用组件时,内容分发API至关重要.